



Un Survol des Méthodes Formelles pour Java CardTM

Davy ROUILLARD

d.rouillard@serma.com

Damien SAUVERON

sauveron@labri.fr

<http://dept-info.labri.u-bordeaux.fr/~sauveron>

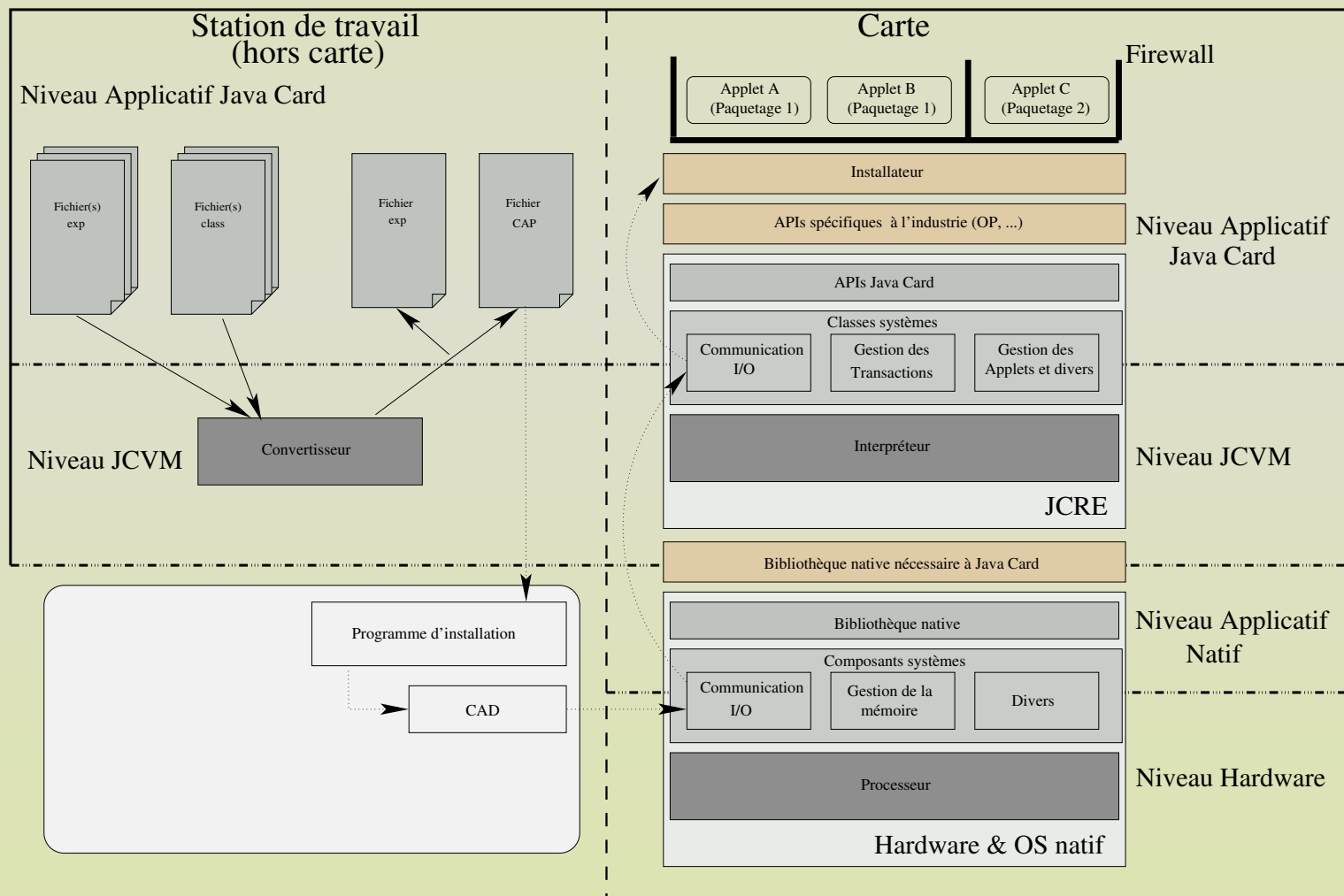
Plan

L'architecture Java Card : vue générale des aspects ciblés.

Projets “*orientés plate-forme*” : prouver des propriétés liées au langage et la JCVM (typage, gestion mémoire, BCV).

Projets “*orientés applications*” : prouver qu'un programme particulier satisfait des propriétés données (invariants sur les données, séquençement des transactions).

L'architecture Java Card



Projets “*orientés plate-forme*”

- ☞ Le projet *CertiCartes* (Inria Sophia-Antipolis : G. Barthe, G. Dufay, B. Serpette...)
 - Formalisation du langage de bytecode en *Coq*.
- ☞ Le projet *Bali* (Münich : G. Klein, T.Nipkow,...)
 - Formalisation du langage JavaCard en *Isabelle*.

Ces projets sont inclus dans *VerifiCard*.

(Université de Nijmegen, INRIA, Universités de Munich et Kaiserslautern, Swedish Institute of Computer Science, SchlumbergerSema.)

Bali et *CertiCartes* s'appuient sur un assistant de preuve (logique d'ordre supérieur).

- ➡ Définition de types et de constantes : type primitif, accès, classe, methode, programme ...
- ➡ Preuve de lemmes.
- ➡ Des techniques pour prouver plus facilement des propriétés : règles de réécritures, règles de raisonnement.

Le projet Certicard

Contient une formalisation de la plate-forme au niveau **bytecode**.

- ➡ formalisation d'un programme (format CAP).
- ➡ formalisation de la machine virtuelle embarquée.
- ➡ formalisation des instructions et leur sémantique (opérationnelle).

Formalisation d'un programme

```
Record jcp program : Set := {
  classes      : (list Class);
  methods      : (list Method);
  interfaces   : (list Interface)
}.

Record Interface : Set := {
  statusshareable : bool;
  super_int       : (list cap_interf_idx);
  Int_id         : cap_interf_idx
}.

Record Method : Set := {
  signature     : (signature_type * type);
  local        : nat;
  bytecode     : (list Instruction);
  statusstatic  : bool;
  handler_list : (list handler_type);
  method_id    : cap_method_idx;
  owner        : cap_class_idx
}.
```

Formalisation de la JCVM

```
Definition jcvm_state := static_heap * heap * stack .
```

```
Inductive obj : Set :=  
  Instance : type_instance -> obj |  
  Array     : type_array -> obj .
```

```
Definition heap := (list obj) .
```

```
Definition static_heap := (list valu) .
```

```
Definition stack := (list frame) .
```

Formalisation des instructions

```
jcvm_state * operands -> returned_state
```

```
Inductive returned_state : Set :=
```

```
Normal      :          jcvm_state -> returned_state |
```

```
Abnormal   : xcpt -> jcvm_state -> returned_state .
```

```
Definition NOP := [state : jcvm_state]
```

```
Cases state of
```

```
(sh, (hp, ((cons h lf) as s))) =>
```

```
  (update_frame (update_pc (S (p_count h)) h) state) |
```

```
_ => (AbortCode state_error state)
```

```
end.
```

Application de Certicard

Description et vérification d'un vérificateur de bytecode (BCV).

A démontrer : *A tout point du programme, les variables locales et les opérandes de pile sont correctement typées pour l'instruction à exécuter.*

Ce qui est prouvé :

- ☞ Le BCV termine.
- ☞ Si la vérification de bytecodes est un succès → `AbortCode` ne sera pas appelée.

Application de *Bali*

```

package java_lang

public interface HasFoo {
    public Base foo(Base z);
}

public class Base implements HasFoo {
    static boolean arr [] = new boolean [2];
    public HasFoo vee;
    public Base foo(Base z) {
        return z;
    }
}

public class Ext extends Base {
    public int vee;
    public Ext foo(Base z) {
        ((Ext)z).vee = 1;
        return null;
    }
}

```

```

BaseCl :: class
"BaseCl == (|access=Public,
    cfields=[(arr, (|access=Public,static=True ,type=PrimT Boolean. []|)),
        (vee, (|access=Public,static=False,type=Iface HasFoo |))],
    methods=[Base_foo],
    init=Expr(arr_viewed_from Base Base
        :=New (PrimT Boolean)[Lit (Intg 2)]),
    super=Object,
    superIfs=[HasFoo|])"

```

Exemples de propriétés

```

package java_lang

public interface HasFoo {
    public Base foo(Base z);
}

public class Base implements HasFoo {
    static boolean arr[] = new boolean[2];
    public HasFoo vee;
    public Base foo(Base z) {
        return z;
    }
}

public class Ext extends Base {
    public int vee;
    public Ext foo(Base z) {
        ((Ext)z).vee = 1;
        return null;
    }
}

```

1. “*La seule interface est HasFoo*”
2. “*foo est déclarée dans Ext*”
3. “*Ext.foo surcharge Base.foo*”

A quoi cela peut-il donc servir ?

➡ *Évaluation de plate-forme* : prouver la correction de certains aspects (BCV, firewall, transaction) en établissant une abstraction avec la formalisation de référence.

⇒ complexe à mettre en oeuvre sans outil (projet Jakarta)

➡ *Évaluation d'application* :

- un compilateur `.cap` → `jcprogram` est fourni par le projet Jakarta.
- des outils de traduction *spécification* → *théories Isabelle/Coq*

⇒ reste à identifier les propriétés à vérifier et le niveau le plus approprié :

$solde \geq 0$; $min \leq val \leq max$; vérification de type énuméré.

Projets orientés “*applications*”

- ➡ Extended Static Checker ESC/Java (Compaq System Research Center)
vérifie à la compilation les assertions ESC/Java ou JML
- ➡ JML : Java Modeling Language (Université de l’Iowa)
- ➡ compilateur *LOOP* (Université de Nijmegen) traduit le code annoté en
JML \longrightarrow PVS ou Isabelle).
- ➡ projet *Bandera*
 - un langage de spécification des propriétés (logique temporelle)
 - des outils pour extraire un modèle à état fini à partir des programmes.
 - un langage d’abstraction et un moteur d’abstraction.
- ➡ *JavaPathFinder*.

JML : Java Modeling Language

- ➡ Développé par l'université de l'état de l'Iowa.
- ➡ Il permet d'exprimer des assertions à inclure dans le code Java en spécifiant, par exemple :
 - les pré-conditions et post-conditions,
 - les invariants,
 - etc.
- ➡ Se rapproche de Eiffel (vérification à l'exécution pour le *debug*) et de l'approche "Design by Contract".
- ➡ Super ensemble de ESC/Java.

Utilisation = annotation du source Java ou Java Card précédée de `//@` ou entourée par `/*@` et `@*/` directement dans le `.java` ou dans un fichier `.jml`

Préconditions et Postconditions en JML

```
/*@ normal_behavior  
  @   requires: <precondition>;  
  @   ensures: <postcondition>;  
  @*/
```

Existence des quantificateurs

- ☞ universels (`\forall`)
- ☞ existentiels (`\exists`)

Traitement des exceptions

```
/*@ behavior
  @   requires: <precondition>;
  @   ensures:  <postcondition>;
  @   signals:  (Exception1) <condition1>;
  .
  .
  .
  @   signals:  (Exceptionn) <conditionn>;
@*/
```

normal_behavior = behavior avec
signals: (java.lang.Exception) false;

invariant et modifiable

Pour la classe APDU (Application Protocol Data Unit), voici un invariant

```
/*@ invariant: buffer != null  
   @           && buffer.length == APDU.BUFFERSIZE;  
   @*/
```

modifiable:x spécifie qu'une méthode peut changer seulement le champ x.

modifiable

```
public static native byte arrayCompare (byte [] src , short srcOff ,
                                         byte [] dest , short destOff ,
                                         short length)

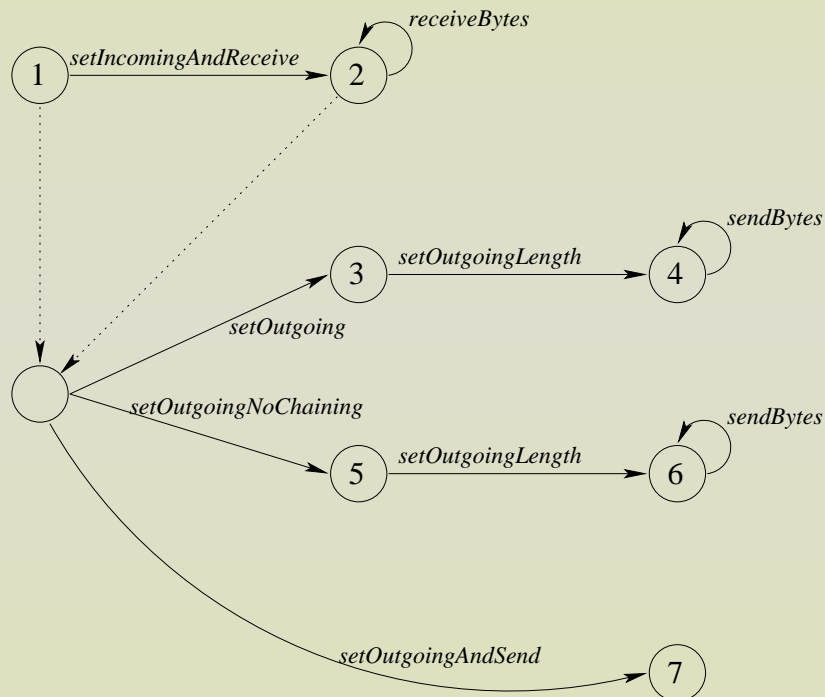
throws ArrayIndexOutOfBoundsException ,
       NullPointerException ;

/*@ normal_behavior
   @
   @   requires : src != null && dest != null &&
   @             srcOff >= 0 && destOff >= 0 && length >= 0 &&
   @             srcOff + length <= src.length &&
   @             destOff + length <= dest.length ;
   @
   @   modifiable : \ nothing ;
   @
   @   ensures : true
   @*/
```

modifiable

```
public static native short arrayCopy(byte [] src , short srcOff ,
                                     byte [] dest , short destOff ,
                                     short length)
throws ArrayIndexOutOfBoundsException ,
       NullPointerException , TransactionException ;
/*@ behavior
@
@   requires : src != null && dest != null &&
@             srcOff >= 0 && destOff >= 0 && length >= 0 &&
@             srcOff + length <= src.length &&
@             destOff + length <= dest.length ;
@
@   modifiable : dest [destOff.. destOff+length - 1];
@
@   ensures : true ;
@
@   signals : ( TransactionException ) true ;
@*/
```

Exemple



```

//@ public model int _APDU_state;
//@ public invariant: 1 <= _APDU_state &&& _APDU_state <= 7;
//@ public model int _Lr;

```

Exemple

```
public short setIncomingAndReceive () throws APDUException ;
/*@ public behavior
   @
   @   requires : _APDU_state == 1;
   @
   @   modifiable : _APDU_state, buffer[5..5+\result-1];
   @
   @   ensures : _APDU_state == 2 &&&
   @             (* data received in buffer[5..5+\result-1] *);
   @
   @   signals : (APDUException) true;
   @*/
```

Exemple

```

public short receiveBytes(short bOff) throws APDUException;
/*@ public behavior
   @
   @   requires: _APDU_state == 2 &&& 0 <= bOff &&&
   @           bOff+getInBlockSize() <= BUFFERSIZE;
   @
   @   modifiable: _APDU_state, buffer[bOff..bOff+\result - 1];
   @
   @   ensures: _APDU_state == 2 &&&
   @           0 <= \result &&&
   @           bOff+\result <= BUFFERSIZE &&&
   @           (* data received in buffer[bOff..bOff+\result - 1] *);
   @
   @   signals: (APDUException) true;
   @*/

```

Exemple

```
public short setOutgoing() throws APDUException;  
/*@ public behavior  
  @  
  @   requires: _APDU_state == 1 || _APDU_state == 2;  
  @  
  @   modifiable: _APDU_state;  
  @  
  @   ensures: _APDU_state == 3;  
  @  
  @   signals: (APDUException) true;  
@*/
```

Exemple

```
public void setOutgoingLength(short len) throws APDUException;
/*@ public behavior
   @
   @   requires:  (_APDU_state == 3 && 0 <= len && len <= 256)
   @             || (_APDU_state == 5 &&
   @                 0 <= len && len <= getOutBlockSize() - 2);
   @
   @   modifiable: _APDU_state, _Lr;
   @
   @   ensures:  _APDU_state == \old(_APDU_state)+1 &&
   @            _Lr == len;
   @
   @   signals:  (APDUException) true;
  @*/
```

Résultats sur l'ADPU

- ➡ A permis de redécouvrir des idées de design de la classe qui avaient disparues dans les spécifications informelles.
- ➡ A permis de lever des problèmes dans l'implémentation de référence.
- ➡ A permis de rendre plus compréhensible les spécifications informelles.

Annexe

```
Record frame : Set := {
  (* operand stack *)
  opstack      : (list valu);

  (* local variables *)
  locvars     : (list valu);

  (* location of the method *)
  method_loc  : cap_method_idx;

  (* context information *)
  context_ref : Package;

  (* whether the method is being analyzed *)
  analyzed    : bool;

  (* program counter *)
  p_count     : bytecode_idx
}.
```